Solving Partial Constraint Satisfaction Problems with Tree Decomposition

Arie M. C. A. Koster
Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Takustraße 7, D-14195 Berlin-Dahlem, Germany

Stan P. M. van Hoesel and Antoon W. J. Kolen
Department of Quantitative Economics, Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands

In this paper, we describe a computational study to solve hard partial constraint satisfaction problems (PCSPs) to optimality. The PCSP is a general class of problems that contains a diversity of problems, such as generalized subgraph problems, MAX-SAT, Boolean quadratic programs, and assignment problems like coloring and frequency planning. We present a dynamic programming algorithm that solves PCSPs based on the structure (tree decomposition) of the underlying constraint graph. With the use of dominance and bounding techniques, we are able to solve small and medium-size instances of the problem to optimality and to obtain good lower bounds for large-size instances within reasonable time and memory limits. © 2002 Wiley Periodicals, Inc.

Keywords: tree decomposition; partial constraint satisfaction; frequency assignment; MAX-SAT; dynamic programming

1. INTRODUCTION

The Partial Constraint Satisfaction Problem (PCSP) consists of a set of decision variables $v \in V$ that have to be assigned exactly one element (value) from a set of feasible values called the *domain* D_v. Each value has a penalty associated, incurred when selected. Moreover, for some pairs of variables $\{v, w\}$, certain combinations of values are also penalized. The objective of the problem is to assign values to all variables so as to minimize the total penalty incurred. If a solution without penalty is searched for, the problem is known as the Constraint Satisfaction Problem (CSP).

In this paper, we study the PCSP with binary relations, that is, all subsets involve only two variables. Let the set $E \subseteq V \times V$ represent all these variable pairs for which some combination of values incurs a nonzero penalty. Then, the problem can be modeled on an auxiliary graph, the *constraint graph,* where the vertices correspond to the decision variables, and the edges, to the pairs in E. Introducing the variables $y(v, d_v)$ for all $v \in V$, $d_v \in D_v$ as

$$y(v, d_v) = \begin{cases} 1 & \text{if } d_v \in D_v \text{ is selected} \\ 0 & \text{otherwise,} \end{cases}$$

we obtain the binary quadratic program

$$\min \sum_{\{v,w\} \in E} \sum_{d_v \in D_v} \sum_{d_w \in D_w} p(v, d_v, w, d_w) y(v, d_v) y(w, d_w) + \sum_{v \in V} \sum_{d_v \in D_v} q(v, d_v) y(v, d_v) \qquad (1)$$

$$\text{s.t.} \quad \sum_{d_v \in D_v} y(v, d_v) = 1 \qquad \forall\, v \in V \qquad (2)$$

$$y(v, d_v) \in \{0, 1\} \qquad \forall\, v \in V,\ d_v \in D_v. \qquad (3)$$

Here, the functions p and q are called the edge penalty function and the vertex penalty function, respectively. Note that this program can be linearized fairly easily introducing the variables $z(v, d_v, w, d_w) = y(v, d_v) y(w, d_w)$ (cf. Padberg [16]).

The PCSP has a wide range of applications. A fairly direct application is the graph coloring problem, where we are given a graph $G = (V, E)$ and k colors that can be assigned to the vertices. This assignment should be such that the number of edges for which the end-vertices have the same color is minimized. There are no penalties involved with the assignment of a color to a vertex, but for each connected pair of vertices, a $k \times k$ unit penalty matrix is given. The graph is k colorable if and only if an assignment without penalty exists. A closely related problem, which actually inspired this research, is a particular formulation of

Received July 1, 2000; accepted July 1, 2002
Correspondence to: A. M. C. A.. Koster; e-mail: koster@zib.de

the Minimum Interference Frequency Assignment Problem (MI-FAP), where antennae are to be assigned one frequency each, taken from a predetermined set of frequencies. For each antenna, certain frequencies are favored over others, and, therefore, there are penalties introduced by assigning frequencies to vertices. Moreover, geographically close antennae may incur interference with one another if close frequencies are assigned to them. Therefore, certain combinations of frequencies are penalized. In the literature, interference is treated (with a few exceptions) as a relation between pairs of antennae. Hence, the MI-FAP can be modeled as a binary PCSP. Moreover, the MI-FAP is a fairly general model of the FAP as many versions can be formulated as a MI-FAP, for instance, the Minimum Span FAP and the Maximum Service FAP; see [3] and *FAP web* [8] for more information. Typically, the penalty matrices resulting from frequency assignment have a diagonal structure. The MI-FAP instances that motivated this study (the so-called radio-link-frequency assignment problems of the CALMA project [2]) have some specific characteristics resulting in more general penalty matrices (see Section 5.2).

Another problem that has an elegant translation to the PCSP is MAX-SAT; see Koster et al. [14]. Finally, many well-known optimization problems on graphs can be transformed to the PCSP. The generalized subgraph problem (for instance, the generalized TSP), as introduced in Feremans [9], is defined as follows: Partition the set of vertices into a number of subsets. Now, select exactly (at most) one vertex in each subset in such a way that the solution on the induced instance is optimal overall.

The $\mathcal{NP}$-hardness of the PCSP with domain sizes at least 3 follows from a reduction from the 3-coloring problem on graphs (cf. [10]). In Koster et al. [14], it was proved, with a reduction from MAX-SAT, that the PCSP is already $\mathcal{NP}$-hard if all domains have size 2. Computational experiments affirm these results in practical settings. In Koster et al. [14], the polyhedral approach is applied with limited success.

In this paper, a lesser-known combinatorial optimization technique is applied to solve PCSPs. Here, we make use of the structure of the constraint graph. In case the graph is *treelike,* we may solve the problem by using a *tree decomposition* of the constraint graph with a small *treewidth.* The notions of treewidth and tree decomposition were introduced by Robertson and Seymour [17] in their fundamental work on graph minors. Besides the major role they play in graph theory, many $\mathcal{NP}$-hard problems on graphs have been shown to be solvable in polynomial (linear) time on graphs with a bounded treewidth (see Bodlaender [5] for an overview). So far, these results have been considered of theoretical interest only. In this paper, we show that this technique is of practical importance as well. Together with some processing techniques, we solve instances of PCSP which could not be solved with other techniques. In addition, with an iterative algorithm that uses the tree decomposition algorithm as a subroutine, lower bounds can be computed for instances that are too large to be solved to optimality.

This paper is organized as follows: In Section 2, we introduce the solution methodology applied in this paper. Tree decompositions and treewidth are defined, a heuristic to find a tree decomposition is presented, and a description of the dynamic programming algorithm based on the tree decomposition of the constraint graph is given. The quality of the algorithm can be improved with the use of (pre)processing techniques, which are described in Section 3. We present the iterative extension of the algorithm that provides lower bounds for the original problem in Section 4. Computational results for MAX-SAT and frequency assignment instances are presented in Section 5.

2. SOLUTION METHODOLOGY

In this section, we describe the solution technique applied in this paper. The method is based on the assumption that the underlying graph structure of the PCSP admits a decomposition with a small so-called treewidth. This allows for a dynamic programming algorithm that solves PCSP in polynomial time (given the treewidth as a constant). In Section 2.1, we introduce tree decompositions and treewidth. Next, we describe in Section 2.2 a heuristic to compute a tree decomposition with a small treewidth. The dynamic programming algorithm is the topic of Section 2.3.

Throughout this paper, we use the standard graph theory terminology as follows: Let $N(v) = \{w \in V : \{v, w\} \in E\}$ denote the set of vertices adjacent to $v \in V$, the neighbors of v, and for set $S \subseteq V$, let $N(S) = \{w \in V \backslash S : \exists_{v \in S}\{v, w\} \in E\}$ be the neighbors of the vertices in S, S excluded. Moreover, let $\delta(S, T)$ denote the set of all edges between the vertices in $S \subseteq V$ and $T \subseteq V \backslash S$, that is, $\delta(S, T) = \{\{v, w\} \in E : v \in S, w \in T\}$. We use $\delta(S)$ as short version of $\delta(S, V \backslash S)$. With $E[S]$, we denote all edges with both vertices in S, that is, $E[S] = \delta(S, S)$. By $G[S] = (S, E[S])$, we denote the subgraph of $G = (V, E)$ induced by S.

2.1. Tree Decomposition and Treewidth

Robertson and Seymour developed the notion of tree decomposition in [17]. A *tree decomposition* $T = (I, F)$ of a graph $G = (V, E)$ is a graph itself, where the nodes are induced subgraphs of G, such that each vertex/edge of G is in at least one subgraph. The edges of the tree decomposition should be chosen such that T is a tree and, moreover, that the nodes of T containing an arbitrary vertex of G induce a connected component in T. The *width* of a tree decomposition is the maximum cardinality of the subgraphs minus one. Formally,

Definition 2.1 (Robertson and Seymour [17])**.** *Let $G = (V, E)$ be a graph. A tree decomposition is a pair $(T, \mathcal{X})$, where $T = (I, F)$ is a tree with nodes I and edges F and $\mathcal{X} = \{X_i : i \in I\}$ is a family of subsets of V, one for each node of T, such that*

$$(i)\ \cup_{i \in I} X_i = V,$$

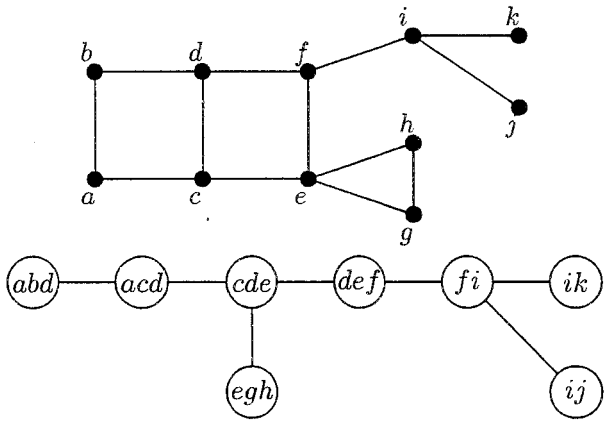


FIG. 1. Example of a graph and a tree decomposition with width 2.

(*ii*) *For every edge* $\{v, w\} \in E$, *there is a node* $i \in I$ *with* $v \in X_i$ *as well as* $w \in X_i$, *and*

(*iii*) *For all* $v \in V$, *the set of nodes* $\{i \in I : v \in X_i\}$ *is connected in* T.

The width of a tree decomposition is $\max_{i \in I} |X_i| - 1$. *The treewidth of a graph* G *is the minimum width over all possible tree decompositions of* G.

In Figure 1, a graph and its optimal tree decomposition are given. The width of this decomposition is 2. Note that a connected graph has treewidth 1 if and only if it is a tree.

2.2. Construction of a Tree Decomposition

The computation of a tree decomposition with minimal width is $\mathcal{NP}$-hard [4]. However, for constant k, the decision whether the treewidth is at most k can be solved in linear time [6]. This algorithm is exponential in k and therefore impractical for graphs with larger treewidth. Therefore, we present in this section a heuristic to compute a tree decomposition. The idea behind the heuristic is based on separating vertex sets. In a tree decomposition T, all internal nodes of T separate G in at least two components. The algorithm aims at decreasing the cardinality of the nodes in a given tree decomposition iteratively. We try to replace a node in an existing tree decomposition by a number of new nodes for which the maximum cardinality is smaller than is the cardinality of the original node. To achieve this goal, we search for small separating vertex sets. Note that finding a separating vertex set with minimum cardinality is a min-cut problem. Hence, it can be found with standard network flow techniques (see [13]).

The heuristic starts with the trivial tree decomposition in which we have one node corresponding to the whole graph. During the process, we have a tree decomposition $(T, \mathcal{X})$ with $T = (I, F)$. We select the node $i \in I$ with $|X_i|$ maximum. This node is replaced by $m + 1$ nodes $i_0, \ldots, i_m$ with vertex sets $X_{i_0}, \ldots, X_{i_m}$. The nodes $i_1, \ldots, i_m$ are all connected to i_0. Each node $k \in N(i)$ is connected to exactly one node $j \in \{i_0, \ldots, i_m\}$, such that all conditions of a tree decomposition are satisfied again. The sets $X_{i_0}, \ldots, X_{i_m}$ are defined as follows: We construct a graph $G_i = (V_i, E_i)$ that consists of the induced subgraph $G[X_i]$ and the additional edges $\cup_{k \in N(i)} C(X_i \cap X_k)$, where $C(X)$ denotes a complete graph on the vertices X [i.e., $C(X)$ is a clique]. If G_i is a complete graph, then $X_{i_0} := X_i$ and $m = 0$, that is, we do not change the tree decomposition. If G_i is not a clique, then we identify a minimum separating vertex set $S \subset V_i$. Let $Y_{i_1}, \ldots, Y_{i_m}$ be the vertex sets of the $m \geq 2$ components of $G_i[V_i \backslash S]$. We define $X_{i_0} := S$, and $X_{i_j} := Y_{i_j} \cup S$ for all $j \in 1, \ldots, m$. A set X_k with $k \neq i$ has a nonempty intersection with at most one set Y_{i_j}, $j = 1, \ldots, m$: Let $v, w \in X_i \cap X_k$; then, $\{v, w\} \in C(X_i \cap X_k) \subset E_i$, which implies that v and w cannot be separated by S. So, either $v, w \in S$ or $v, w \in Y_{i_j} \cup S$ for only one $j \in \{1, \ldots, m\}$. Therefore, we connect each neighbor $k \in N(i)$ with the node i_j, $j \in \{1, \ldots, m\}$, for which the intersection of X_k and Y_{i_j} is nonempty; in case there is no such i_j, we connect k with i_0. As a consequence, the new construction is a tree again (see Fig. 2). In the new tree, the conditions for a valid tree decomposition again hold. Since $\cup_{j=0}^{m} X_{i_j} = (\cup_{j=0}^{m} Y_{i_j}) \cup S = X_i$, condition (i) is satisfied. To satisfy condition (ii), we have to prove that, for each edge $\{v, w\} \in E[X_i]$, one of the new vertex sets $X_{i_0}, \ldots, X_{i_m}$ contains both vertices. If $v, w \in S$, then this is trivially true. Otherwise, suppose that $v \in Y_{i_j}$ for some $j \in \{1, \ldots, m\}$. If $w \in Y_{i_k}$, $k \neq j$, then S does not separate Y_{i_j} and Y_{i_k}, a contradiction. Thus, $w \in Y_{i_j} \cup S = X_{i_j}$. Condition (iii) states that all nodes in the tree that contain the same vertex v must form a subtree. We only need to check this for $v \in X_i$. If $v \in S$, then v is contained in all new nodes and the condition is trivially satisfied. Otherwise, let $v \in Y_{i_j}$ for some $j \in \{1, \ldots, m\}$. By construction, nodes $k \in N(i)$ and i_j are connected if X_k and Y_{i_j} intersect. Hence, all nodes that contain v form a subtree again.

Note that if G_i is not a clique then there exist vertices $v, w \in X_i$ with $\{v, w\} \notin E_i$. Thus, $S = X_i \backslash \{v, w\}$ separates G_i into two components: $Y_{i_1} = \{v\}$ and $Y_{i_2} = \{w\}$. So, $\max\{|Y_{i_1} \cup S|, |Y_{i_2} \cup S|\} = |X_i| - 1 < |X_i|$. As a consequence, the width of the tree decomposition may decrease. Figure 3 shows the heuristic in a flowchart.

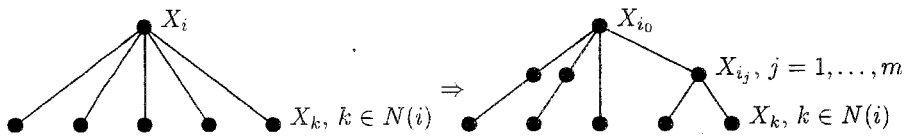


FIG. 2. Improvement step of a tree decomposition.

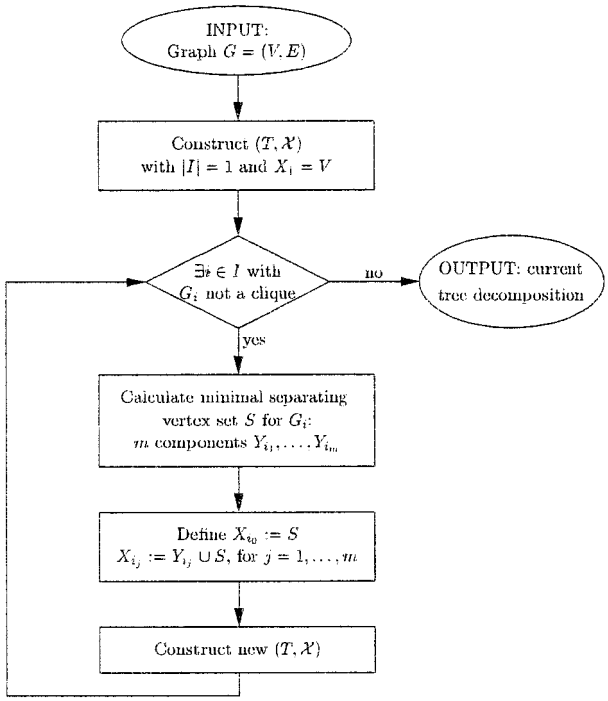


FIG. 3. Heuristic for construction of a tree decomposition.

2.3. Dynamic Programming Algorithm

Similar to many other $\mathcal{NP}$-hard problems based on graphs, the PCSP can be solved in polynomial time for graphs of bounded treewidth. The algorithm is based on the following idea: Let $S \subset V$ be a separating vertex set of G with $G[V \backslash S] = G[V_1] \cup G[V_2]$. Then, the optimal assignment in V_1 (or V_2) only depends on the assignment in S. So, given an assignment of S, the problem decomposes into two independent PCSPs on $G[V_1]$ and $G[V_2]$, which can be solved separately. This idea can be formulated as a dynamic programming algorithm using a tree decomposition $(T, \mathcal{X})$ of the graph. For every internal node $i \in I$, X_i is a separating vertex set, which implies that, given an assignment for X_i, the PCSP decomposes into smaller PCSPs for every branch in the tree.

To describe the algorithm in more detail, we need some additional notation: For the remainder of the paper, we assume that the tree is rooted and binary. Let $Y_i = \{ v \in V : \exists j \in I, v \in X_j \text{ and } j \text{ descendant of } i \}$ denote the set of vertices that is represented by the subtree rooted at node i. Given a subset $S \subseteq V$, we denote with $d_S = (d_v)_{v \in S}$ an assignment of domain elements $d_v \in D_v$ for every vertex $v \in S$. Similarly, D_S denotes the complete set of assignments for a given set S.

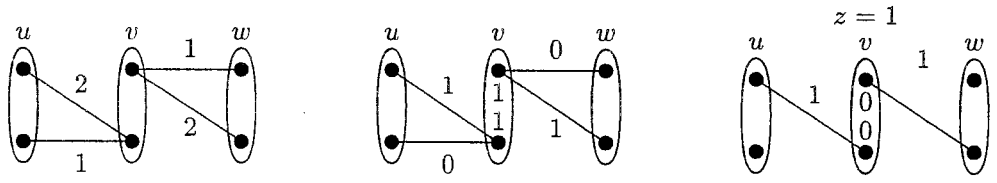
Now, in a bottom-to-top way, the dynamic programming algorithm computes for every node $i \in I$ all assignments D_{Y_i} for the subset Y_i. Starting with a leaf $i \in I$ of the tree, the algorithm stores all assignments for the vertices in X_i. The computation of all assignments takes $\mathcal{O}(\Pi_{v \in X_i} |D_v|) = \mathcal{O}(d^{|X_i|})$ time, where $d = \max_{v \in V} |D_v|$. Next, given all assignments for two nodes $j, k \in I$ with common predecessor $i \in I$, we can compute all assignments D_{Y_i} by combining every assignment of Y_j, every assignment of Y_k that has the same assignment for the vertices in $X_j \cap X_k$, and every assignment of domain elements to the vertices in $X_i \backslash (X_j \cup X_k)$. However, since X_i is a separating vertex set in the graph, we do not have to store all assignments for the vertices in Y_i, but only the assignments that differ for the vertices in X_i. For an assignment of the vertices in X_i, we only have to store the best assignment for the vertices in $Y_i \backslash X_i$. In other words, we have to store, at most, $\Pi_{v \in X_i} |D_v|$ assignments for node $i \in I$ instead of $\Pi_{v \in Y_i} |D_v|$ assignments to obtain the overall optimal solution. The computation of these assignments can be done in $\mathcal{O}(\Pi_{v \in X_i \cup X_j \cup X_k} |D_v|) = \mathcal{O}(d^{|X_i| + |X_j| + |X_k|})$ time. Finally, for the root node $r \in I$ of the tree T, $Y_r = V$, and so we only have to store one solution which gives the desired optimal solution for the problem. The overall computation time of this algorithm is given by $\mathcal{O}(nd^{3k})$, where k is the width of the tree decomposition $(T, \mathcal{X})$ of G that is used. So, for graphs with a treewidth bounded by a constant k, this algorithm solves the PCSP in time polynomial in n and d, but exponential in k.

3. REDUCTION TECHNIQUES

The performance of the dynamic programming algorithm highly relies on additional techniques to reduce the size of the sets of assignments. We describe two types of reduction techniques, namely, bounding and dominance. We present simple techniques that are quickly performed, but also very complex techniques which are to be processed only when necessary. In applying these techniques, there is always a delicate balance between speed and effectiveness. Nevertheless, all techniques are presented in a unified way. All techniques are based on the following straightforward paradigm for extending partial feasible solutions:

> *A partial feasible solution can be extended to an optimal solution* ***only if*** *the extension itself is the best possible with respect to the partial feasible solution. In other words, if a partial feasible solution is not extended optimally, the resulting feasible solution is certainly not optimal.*

In the first subsection, we use this paradigm directly to remove vertices or to replace them by edges. In Subsection 3.2, we present a penalty shifting procedure, which is used mainly to obtain lower bounds on the value of the instances, but can sometimes be used to remove edges from the constraint graph as well. In Subsection 3.3, we present techniques to remove values from the domains of vertices and to remove nonoptimal partial feasible solutions. This is done in two ways: by using upper-bounding techniques and by using dominance criteria. In Subsection 3.4, we conclude with a description of how these techniques are applied in practice.



(a) original (b) shift to vertex v (c) shift to objective

FIG. 4. Example of shifting penalties.

3.1. Constraint Graph Reduction

In this subsection, we describe how we can remove vertices $v \in V$ with $|N(v)| \leq 2$ from G. First, consider a vertex $v \in V$ with one neighbor, say w. Then, if in a partial feasible solution, w is assigned a value d_w^*, the optimal choice for v is given by $\arg\min_{d_v \in D_v}\{q(v, d_v) + p(v, d_v, w, d_w^*)\}$. Although d_w^* may differ among all partial solutions, we can determine the best extension of any partial feasible solution beforehand by, for all $d_w \in D_w$, computing the value

$$q'(w, d_w) = \min_{d_v \in D_v}\{q(v, d_v) + p(v, d_v, w, d_w)\}$$

and subsequently adding $q'(w, d_w)$ to $q(w, d_w)$. This, in effect, adds to each d_w the optimal choice in D_v at the beginning of the algorithm, allowing us to remove the vertex v and the edge $\{v, w\}$ from the instance.

We can generalize this idea to vertices with degree two as follows: Let v be such a vertex, and let $N(v) = \{u, w\}$. Then, for a partial solution with d_u^* and d_w^* selected for u and w, respectively, the optimal choice for v is $d_v^* = \arg\min_{d_v \in D_v}\{p(u, d_u^*, v, d_v) + q(v, d_v) + p(v, d_v, w, d_w^*)\}$. Again, we can do this beforehand by, for all $d_u \in D_u$, $d_w \in D_w$, computing the value

$$p'(u, d_u, w, d_w) = \min_{d_v \in D_v}\{p(u, d_u, v, d_v) + q(v, d_v) + p(v, d_v, w, d_w)\}$$

and subsequently adding $p'(u, d_u, w, d_w)$ to $p(u, d_u, w, d_w)$. This, in effect, adds to each combination $\{d_u, d_w\}$ the optimal choice in D_v, allowing us to remove the vertex v and its two incident edges from the instance. Note that possibly the edge $\{u, w\}$ may have to be inserted in the constraint graph.

We can repeat the reduction process until all vertices with degree at most two are removed.

3.2. Penalty Shifting: Lower Bounding

In this subsection, we present a technique to obtain a lower bound on the optimal value of the instances by shifting penalties from edges to vertices and back and from vertices to the objective and back.

If for an edge $\{v, w\} \in E$ we have penalties with the property that for some $d_v^* \in D_v$, $p(v, d_v^*, w, d_w) > 0$ for all $d_w \in D_w$, then we can decrease these penalties and simultaneously increase $q(v, d_v^*)$ by the same amount. The same procedure works on vertices. Suppose that we have a positive penalty $q(v, d_v)$ for all $d_v \in D_v$. Then, by (2), we can decrease the penalty $q(v, d_v)$ by the minimum vertex penalty and add the same value to the objective. The condition that all penalties should be nonnegative is not really crucial, but allows us to maintain a lower bound on the objective value. Figure 4 illustrates this technique. We have three vertices, each with two domain elements. Nonzero edge penalties are indicated beside the edges [Fig. 4(a)]. By the described procedure, we first can decrease the edge penalties by increasing the vertex penalties of v [cf. Fig. 4(b)]. Next, we can subtract 1 from the vertex penalties of v and add this value to the constant part of the objective [Fig. 4(c)].

3.3. Domain Reduction

In this section, we present methods to reduce the number of partial feasible solutions by upper bounding (Section 3.3.1) and dominance (Section 3.3.2).

3.3.1. Upper Bounding. Upper bounding in its simplest form is performed on vertices as follows: Consider a vertex v and its neighbors $N(v)$. We want to derive an upper bound $u(v, \delta(v))$ on the total penalty incurred by node v in the optimal solution of the PCSP, that is, an upper bound on the vertex penalty of v and the edge penalties of the edges incident with v.

Consider an arbitrary partial solution $d_{N(v)}^* \in D_{N(v)}$. Then, we compute the value for v with the lowest penalty:

$$P(d_{N(v)}^*) = \min_{d_v \in D_v}\{q(v, d_v) + \sum_{w \in N(v)} p(v, d_v, w, d_w^*)\}.$$

Among all possible choices for $d_{N(v)}^* \in D_{N(v)}$, we take the one with highest penalty, that is,

$$u(v, \delta(v)) = \max_{d_{N(v)}^* \in D_{N(v)}} P(d_{N(v)}^*).$$

Then, the value $u(v, \delta(v))$ is certainly an upper bound on the penalty incurred by an optimal choice of value for v. So, if $q(v, d_v) > u(v, \delta(v))$, then d_v can be removed from the domain D_v. We apply this idea in a preprocessing phase of the dynamic program.

This technique can be generalized to sets of vertices $S \subset V$, instead of single vertices. For arbitrary $S \subset V$, we can compute this upper bound by solving an integer linear program. For all $w \in N(S)$, $d_w \in D_w$, we introduce a binary variable:

$$y(w, d_w) = \begin{cases} 1 & \text{if } d_w \in D_w \text{ is assigned to } w \in N(S) \\ 0 & \text{otherwise.} \end{cases}$$

If the variable z denotes the actual upper bound, the integer linear program becomes

$$\mathrm{u(S, \delta(S))} = \max z \tag{4}$$

$$\text{s.t.} \quad z \le q(S, d_S) + \sum_{w \in N(S)} \sum_{d_w \in D_w} \sum_{v \in N(w) \cap S} \times \; p(v, d_v, w, d_w) y(w, d_w)$$

$$\forall\, d_S \in D_S \tag{5}$$

$$\sum_{d_w \in D_w} y(w, d_w) = 1 \qquad \forall\, w \in N(S) \tag{6}$$

$$y(w, d_w) \in \{0, 1\} \qquad \forall\, w \in N(S),\, d_w \in D_w. \tag{7}$$

Here, $q(S, d_S)$ denotes the total penalty involved in an assignment d_S, that is,

$$q(S, d_S) = \sum_{v \in S} q(v, d_v) + \sum_{\{v, w\} \in E[S]} p(v, d_v, w, d_w).$$

The constraints (6) and (7) enforce that, for each neighbor of S, exactly one value is chosen. For a given choice of values $d^*_{N(S)}$, the right-hand sides of constraints (5) are the penalties incurred with each of the corresponding assignments for S. Thus, an assignment d_S with the smallest penalty determines the highest value z can obtain for the particular choice of values for the neighbors of S. For each possible assignment of values to the neighbors of S, we determine this value. The worst choice of $d^*_{N(S)}$ is the one for which this value is maximal. This choice determines the value of z, that is, $u(S, \delta(S))$.

An assignment $d_S \in D_S$ with $q(S, d_S) > u(S, \delta(S))$ cannot be extended to an optimal complete assignment. Hence, d_S can be removed from the set of assignments D_S. An assignment with $q(S, d_S) \le u(S, \delta(S))$ is called nonredundant.

The main problem with $u(S, \delta(S))$ is that it takes time to compute. It may be preferable to compute the value of some relaxation of (4)–(7). The standard LP-relaxation does not generate really powerful upper bounds. Our choice is therefore to relax (4)–(7) by taking a subset of the constraints (5), that is, a number of partial feasible solutions with low $q(S, d_S)$. In case we restrict ourselves to one good partial solution d^*_S for S, we can solve the relaxed problem by inspection and use this as an upper estimate of $u(S, \delta(S))$:

$$u(S, \delta(S)) \le q(S, d^*_S)$$

$$+ \sum_{w \in N(S)} \sum_{d_w \in D_w} \sum_{v \in N(w) \cap S} \times \; p(v, d^*_v, w, d_w) y(w, d_w)$$

$$= q(S, d^*_S) + \sum_{w \in N(S)} \max_{d_w \in D_w} \sum_{v \in N(w) \cap S} p(v, d^*_v, w, d_w). \tag{8}$$

The upper bound $u(S, \delta(S))$ is especially powerful if the number of edges in the cut-set $\delta(S)$ is small or if the sum of the maximum penalties incurred by the cut-set edges is not too large; see (8). If the upper bound $u(S, \delta(S)) = 0$ for a subset S, then we know that, given any assignment to the vertices $V \backslash S$, the partial solution can be extended to a complete solution without additional penalty. This implies that we can remove the subset S and the edges $\delta(S)$ from the constraint graph.

3.3.2. Dominance. Upper-bounding techniques are a quick way to eliminate the worst domain elements (or partial feasible solutions), but these techniques sometimes only remove a small fraction of the values that are redundant. The upper bounding technique can be advanced by looking at individual domain elements.

Let $v \in V$. Consider $D_{N(v)}$, that is, the partial solutions of $N(v)$. If these partial solutions can all be extended optimally with a value of $D_v \backslash \{d^*_v\}$, then d^*_v is not necessary. Hence, d^*_v can be removed from D_v. We say that d^*_v is *dominated* by the values in $D_v \backslash \{d^*_v\}$. This concept can also be generalized to sets of vertices, similar to the generalization of the upper bounds to sets $S \subset V$. Let d^*_S be an assignment to S; then, d^*_S is dominated by the other nonredundant assignments $D_S \backslash \{d^*_S\}$ if every partial feasible solution of $N(S)$ can be extended to a solution at minimum cost with an assignment of $D_S \backslash \{d^*_S\}$.

The decision problem of whether a partial feasible solution is dominated by the remaining ones can be modeled as an integer linear program in a similar way as in the case for the upper bound $u(S, \delta(S))$ (see [15] for details). Since this formulation has a constraint for all remaining partial feasible solutions, it again is worthwhile to relax some of them to determine dominance faster. In the most extreme form, we compare a partial feasible solution d^*_S with only one d_S. Let $\Delta p(v, d_v, d^*_v, w, d_w) = p(v, d_v, w, d_w) - p(v, d^*_v, w, d_w)$ for all $\{v, w\} \in \delta(S)$, $d_w \in D_w$. If

d_v, d_w	1	2	3	4
1	10	10	0	0
2	10	10	10	0
3	0	10	10	10
4	0	0	10	10

(a) original penalty matrix

d_v, d_w	$\{1,2\}$	$\{3,4\}$
$\{1,2\}$	10	0
$\{3,4\}$	0	10

(b) new penalty matrix

FIG. 5. Example to illustrate the idea behind the iterative algorithm.

$$q(S, d_S) - q(S, d_S^*) + \sum_{\{v,w\}\in\delta(S)} \max_{d_w\in D_w} \Delta p(v, d_v, d_v^*, w, d_w) \le 0, \quad (9)$$

then d_S^* is dominated by d_S.

3.4. Implementation Issues

The above-described techniques are applied in both a preprocessing phase and during the dynamic programming algorithm. In the preprocessing phase, we apply the graph reduction techniques, the penalty shifting, as well as the upper bounding and dominance for single vertices until no further reduction is achieved. During the dynamic programming algorithm, the upper bounding and the simplified dominance test (9) are applied for every computed separating vertex set.

4. ITERATIVE VERSION OF THE DYNAMIC PROGRAMMING ALGORITHM

Both time and memory are sometimes insufficient to solve large instances with the dynamic programming algorithm described in Section 2.3, even if we use the reduction techniques of Section 3. During the execution of the algorithm, the number of nonredundant assignments explodes for these instances. We can point out two reasons: On the one hand, the width of our tree decomposition is too large. On the other hand, the number of domain elements of a vertex is too large. In the latter case, we may make use of the structure of the penalties. In the MI-FAP instances of the CALMA project, for example, the penalty matrices contain large blocks of penalties that are almost equal. In this case, it is *relatively* uncritical to assign either one of the frequencies in such a block. The large differences in total penalty are caused by changing the assigned frequency from one block to another. This structure of the penalty matrices allows for the following approach: Instead of assigning single values to the vertices, identify a subset of values with each vertex. By estimating the vertex and edge penalties for these subsets from below, the optimal value of this newly created PCSP is a lower bound on the optimal value of the original PCSP. The time and memory requirements for solving the newly created PCSP are much smaller, since the domains of the vertices are shrunken substantially.

Let us illustrate this idea with an example depicted in Figure 5. Figure 5(a) shows the penalty matrix for a particular edge. The level of interference on this edge is 10 if the difference between the values (frequencies) is less than 2. If we divide the values into two groups $\{1, 2\}$, and $\{3, 4\}$, we obtain four blocks in the table of edge penalties with (almost) the same values. In most cases, there is no difference between the penalties as long as the pairs of values are in the same block. Therefore, let us construct a new PCSP in which we have to assign either the subset $\{1, 2\}$ or the subset $\{3, 4\}$ to the vertices. The edge penalties in this newly created PCSP are given by the minimum of the values in each block [see Fig. 5(b)]. Solving this substantially smaller problem provides a lower bound on the optimal value of the original problem. The quality of the lower bound depends on the size of the blocks: Many small blocks will provide a better lower bound than will a small number of large blocks. As mentioned before, in applications such as MI-FAP, the block structure of the penalty matrices arises naturally, since the available values for an antenna can be divided into groups of values that are in the same part of the spectrum.

We can extend this idea to an iterative method that provides a sequence of lower bounds for the original instance. The dynamic programming algorithm is used as a subroutine to solve the newly created PCSPs. The iterative algorithm starts with an initial partition of the domains. Solving this PCSP results in a lower bound on the optimal value for the original PCSP. Next, given a solution for this first subproblem, the partition of the domains in subsets is refined in such a way that the newly created PCSP provides a lower bound that is at least as good (but hopefully better) as the previous one. This process of creating PCSPs with refined subsets is repeated until either (i) a solution is found where each assigned subset consists of a single domain element, (ii) a solution whose value is equal to the upper bound, or (iii) time and/or memory requirements prevent us from solving the actual subproblem. For further details on the iterative algorithm in general, and the refinement of the subsets in particular, we refer to the technical report version of this paper [15].

TABLE 1. Computational results for the dynamic programming algorithm (MAX-SAT instances).

Instance	No. variables	No. clauses	$\|V\|$	$\|E\|$	Computed width	CPU time (seconds) Heur.	CPU time (seconds) DP
dubois20	60	160	220	480	3	176.1	142.4
dubois21	63	168	231	504	3	160.2	156.3
dubois22	66	176	242	528	3	186.9	171.5
dubois23	69	184	253	552	3	269.5	190.4
dubois24	72	192	264	576	3	309.2	206.7
dubois25	75	200	275	600	3	354.3	223.4
dubois26	78	208	286	624	3	400.4	242.1
dubois27	81	216	297	648	3	453.2	262.3
dubois28	84	224	308	672	3	508.4	280.7
dubois29	87	232	319	696	3	570.8	302.0
dubois30	90	240	330	720	3	657.3	322.5
dubois50	150	400	550	1200	3	3509.0	894.2
dubois100	300	800	1100	2400	3	35,969.6	3585.0
pret60_40	60	160	220	480	8	176.2	138.0
pret60_60	60	160	220	480	8	131.6	138.6
pret60_75	60	160	220	480	8	131.2	136.2
pret150_40	150	400	550	1200	8	2468.6	10,137.4
pret150_60	150	400	550	1200	8	2465.0	10,180.7
pret150_75	150	400	550	1200	8	2466.6	10,142.6

5. COMPUTATIONAL RESULTS

The approach described in the previous sections was tested on two sets of PCSPs. The first set consisted of 19 MAX-SAT instances taken from the second DIMACS challenge on cliques, colorings, and satisfiability [7]. The second set of instances was taken from the CALMA project [2] on the radio-link-frequency assignment. The algorithms were implemented in C++. We used the callable library of CPLEX, version 4, to solve (integer) linear programming problems (upper bounding, dominance).

5.1. MAX-SAT

Maximum satisfiability (MAX-SAT) problems can be converted to PCSPs [14]. Given a set of variables X and a set of clauses C, a graph with $|X| + |C|$ vertices and $\Sigma_{c \in C} x_c$ edges is constructed, where x_c is the number of variables in clause c. As all considered instances are, in fact, MAX-3-SAT instances, the number of edges equals $3|C|$. The domains of the vertices contain either two elements (variables) or x_c elements (clauses). The two domain elements for a variable are *true* and *false.* Each clause has a domain element for every variable (or its negation) in the clause. The edges connect the clauses with the variables. A variable and a clause are connected if the variable (or its negation) is in the clause. If the variable x is in the clause, the only combination of elements that is penalized is $\{false, x\}$; if the negation of x is in the clause, only the combination $\{true, \bar{x}\}$ is penalized. All penalties are equal (one). Now, k clauses can be satisfied if and only if there exists a solution to the PCSP with value $|C| - k$. So, the problems are equivalent.

Since the domains are pretty small, we do not perform any (pre)processing for the MAX-SAT instances. Also, the iterative version of the algorithm is not performed. Table 1 shows the results for both computing a tree decomposition of the graph (columns "computed width" and "Heur.") and the dynamic programming algorithm (column "DP"). The computations were carried out on a Linux-operated PC with a Pentium III 800 MHz processor and 512 MB of internal memory. The "dubois" instances have such a structure that the treewidth is 3 in all cases, whereas for the "pret" instances tree decompositions of width 8 are computed by the heuristic described in Section 2.2. For the dynamic programming algorithm, only the computation time is reported since the optimal value for all instances equals one, that is, only one clause cannot be satisfied. The results show that the heuristic to compute a tree decomposition is, in almost all cases, the most time-consuming part. Given a decomposition, the time required by the dynamic programming algorithm increases more or less linearly in the number of clauses/variables for the "dubois" instances (except for the instances "dubois50" and "dubois100"). For the "pret" instances, the computation times of the dynamic programming algorithm increase much faster, which can be explained by the fact that the width of the tree decomposition (in this case 8) is part of the exponent of the running time.

5.2. Radio–link–frequency Assignment

Within the CALMA project, 11 instances for minimizing the interference in a military radio-link network are provided. The CELAR instances are real life; the GRAPH instances are randomly generated with the same characteristics as the CELAR instances. An overview of the results

TABLE 2. Statistics and preprocessing results (CALMA instances).

Instance	Before preprocessing			After preprocessing				Computed width	Lower bound
	$\|V\|$	$\|E\|$	$\|D\|$	$\|V\|$	$\|E\|$	$\|D\|$	Fixed		
CELAR 06	100	350	39.9	82	327	39.9	0	11	10
CELAR 07	200	817	39.9	162	764	34.6	0	17	10
CELAR 08	458	1655	39.5	365	1539	39.4	0	18	10
CELAR 09	340	1130	39.5	67	165	35.6	11,391	7	7
CELAR 10	340	1130	39.5	0	0	—	31,516	—	—
GRAPH 05	100	416	37.1	0	0	—	221	—	—
GRAPH 06	200	843	37.7	119	348	16.2	4112	17	5
GRAPH 07	200	843	36.7	0	0	—	4324	—	—
GRAPH 11	340	1425	37.7	340	1425	32.6	2553	104	7
GRAPH 12	340	1255	37.6	61	123	15.3	11,496	4	4
GRAPH 13	458	1877	38.4	456	1874	38.1	8676	133	6

available for these instances can be found at *FAP web* [8] (see also [3]).

Although all (minimum interference) frequency assignment problems can be formulated as PCSPs, the characteristics of radio link frequency assignment make these instances particularly suitable for the tree decomposition approach. The most eye-catching difference between radio-link-frequency assignment and other wireless communication systems is that the connections are established mobile-to-mobile without the interposition of so-called base stations. Hence, we cannot distinguish between an up- and downlink direction of a bidirectional connection and so cannot use separate frequency bands for the directions. The only requirement is that the frequencies assigned to both directions are at a fixed distance. Consequently, the penalty matrices have a far more arbitrary structure in comparison with those from, for example, GSM frequency planning (see [8, 13] for more details). In this section, we solve seven of the 11 instances to optimality and we obtain good lower bounds for the other instances. Before this study, nontrivial lower bounds were only available for two instances.

The solution procedure consists of three steps: (i) instance preprocessing, (ii) computation of a tree decomposition, and (iii) application of the dynamic programming algorithm including the processing techniques. In case time or computer memory is insufficient for computing the optimal solution, the iterative version of the dynamic programming algorithm is applied. The computations are performed on a DEC 2100 A500MP workstation with 128 MB internal memory. We refer to [15] for implementation details.

Table 2 reports on the first two steps of the solution procedure. For all instances, problem statistics before and after preprocessing are reported. We report the number of vertices ($|V|$), the number of edges ($|E|$), and the average number of domain elements ($|D|$). In addition, we report the value that is fixed by the preprocessing phase. The last two columns of the table show the width computed by our heuristic and a lower bound on the treewidth given by the maximum clique size minus one (note that every clique should be in at least one node of the tree).

From Table 2 we can draw the following conclusions: For the instances CELAR 06–08, the graph reduction techniques result in an instance size reduction of roughly 20%. The domain reduction techniques and penalty shifting have only a marginal effect. For the other instances, the combination of graph reduction, penalty shifting, and domain reduction techniques is very successful. In this way, three instances are solved by preprocessing only. For the other instances, the tightness of many constraints resulted in reduced domains and nontrivial lower bounds on the optimal value. The running time of the preprocessing phase is within 1 minute for all instances.

For the second step, Table 2 shows that the gap between the computed width and the lower bound varies from zero for small instances to very large values for the instances GRAPH 11 and GRAPH 13. For these instances, it is not clear which bound is poor. We tried several variants of our heuristic to improve the width of the tree decomposition, but without any success.

Results for the third step and for the iterative algorithm are reported in Table 3. Besides the best lower and upper bound known, and the value fixed by preprocessing, we report on the value obtained by the dynamic programming algorithm and the iterative method, as well as their computation times. These results were obtained with the dynamic programming algorithm without dominance. Experiments with the dominance test did not result in a better performance of the algorithm for these instances. The instances CELAR 09, GRAPH 06, and GRAPH 12 can be solved very efficiently with this method, due to the availability of a tree decomposition with a small width. Instance CELAR 06 takes more than 7.5 hours. The optimal value for all these instances is equal to the best-known upper bound from Kolen [12]. Note that without preprocessing and upper bounding it is impossible to solve these instances due to the large number of partial solutions that have to be kept in the memory. For the same reason, the other instances could not be solved with the dynamic programming algorithm, even with the application of preprocessing and upper bounding.

For these instances, as well as for instance CELAR 06,

TABLE 3. Results of dynamic programming (CALMA instances).

Instance	Best lower bound [1, 11]	Best upper bound [12]	Fixed by preprocessing	Dynamic programming (optimum)	Iterative method (lower bound)	CPU time (seconds)	
						Dynamic programming	Iterative method
CELAR 06	5	3389	0	3389	3388	27,102	9429
CELAR 07	5	343,592	0	—	300,000	—	275,736
CELAR 08	0	262	0	—	87	—	313,168
CELAR 09	14,969	15,571	11,391	15,571	—[a]	23	—[a]
CELAR 10	31,204	31,516	31,516		Solved by preprocessing		
GRAPH 05	—	221	221		Solved by preprocessing		
GRAPH 06	—	4123	4112	4123	—[a]	29	—[a]
GRAPH 07	—	4324	4324		Solved by preprocessing		
GRAPH 11	—	3080	2553	—	—	—	—
GRAPH 12	—	11,827	11,496	11,827	—[a]	11	—[a]
GRAPH 13	—	10,110	8676	—	—	—	—

[a] Iterative method not applied, since problem is solved rapidly by dynamic programming.

we applied the discussed iterative algorithm. For each instance, two runs of the algorithm were carried out with different initial domain partitions. In one, we partition each domain into two subsets and, in the other, into four subsets. The best result is reported in Table 3. For CELAR 06, the algorithm stops since it cannot find any further refinement of the domain partitions. For all other instances, the algorithm stops due again to memory limitations.

The lower bound derived in this way for CELAR 06 is very strong: only one below the optimal value. For both CELAR 07 and CELAR 08, the values are the first nontrivial lower bounds. The gap between lower and upper bounds is closed by, respectively, 87.3 and 33.2%. For the instances GRAPH 11 and GRAPH 13, the width of the tree decomposition is too large to apply the dynamic programming algorithm with any success.

6. CONCLUDING REMARKS

In this paper, we described how the concept of tree decomposition can be used to compute an optimal solution of partial constraint satisfaction problems. From theory, it is well known that many combinatorial optimization problems can be solved in polynomial time if the treewidth of the underlying graph is bounded by a constant. The practical relevance, however, was hard to determine, due to a lack of implementations. In this paper, we report on a computational study to solve partial constraint satisfaction problems with this technique. Many $\mathcal{NP}$-complete problems such as MAX-SAT, k-coloring, and frequency assignment can be easily formulated as PCSPs. Therefore, our method can be used to solve MAX-SAT and frequency-assignment instances. For frequency assignment, this result could only be achieved with the aid of additional reduction techniques, like graph reduction, upper bounding, and dominance. By the introduction of an iterative algorithm that uses a tree decomposition-based dynamic program as a subroutine, the first nontrivial lower bounds could be derived for most of these instances.

In conclusion, these results show that the tree decomposition approach can be an alternative to integer programming, especially in those cases where the latter method fails. To be competitive on a larger class of problems, there is a need for further ideas on how to improve the performance of both the heuristic to construct a tree decomposition (which is a problem independent task) and the dynamic programming algorithm (where problem-specific properties have to be taken into account).

Acknowledgments

The authors would like to thank Rudolf Müller for his suggestions which resulted in the iterative algorithm of Section 4. Moreover, they would like to thank the referees for their time and effort to point out many significant improvements to earlier versions of this paper. This research was carried out while the corresponding author was a Ph.D. student at Maastricht University.

REFERENCES

[1] K.I. Aardal, A. Hipolito, C.P.M. van Hoesel, and B. Jansen, A branch-and-cut algorithm for the frequency assignment problem, Research Memorandum 96/011, Maastricht University, 1996. Available at http://www-edocs.unimaas.nl/abs/rm96011.htm.

[2] K.I. Aardal, C.A.J. Hurkens, J.K. Lenstra, and S.R. Tiourine, Algorithms for radio link frequency assignment: The CALMA project, Oper Res (to appear).

[3] K.I. Aardal, C.P.M. van Hoesel, A.M.C.A. Koster, C. Mannino, and A. Sassano, Models and solution techniques for the frequency assignment problem, ZIB-report 01-40, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2001. Available at http://www.zib.de/PaperWeb/abstracts/ZR-01-40/.

[4] S. Arnborg, D.G. Corneil, and A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J Alg Discr Methods 8 (1987), 277–284.

[5] H.L. Bodlaender, A tourist guide through treewidth, Acta Cybernet 11 (1993), 1–21.

[6] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, SIAM J Comput 25 (1996), 1305–1317.

[7] The second DIMACS implementation challenge: $\mathcal{NP}$-hard problems: Maximum clique, graph coloring, and satisfiability; see http://dimacs.rutgers.edu/Challenges/ or http://mat.gsia.cmu.edu/challenge.html, 1992–1993.

[8] A. Eisenblätter and A.M.C.A. Koster, FAP web—A website devoted to frequency assignment. URL: http://fap.zib.de, 2000.

[9] C. Feremans, Generalized spanning trees and extensions, Ph.D. Thesis, Université Libre de Bruxelles, 2001. Available at http://smg.ulb.ac.be/Theses/Theses.html.

[10] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of $\mathcal{NP}$-completeness, Freeman, New York, 1979.

[11] C.A.J. Hurkens and S.R. Tiourine, Upper and lower bounding techniques for frequency assignment problems, Technical Report COSOR 95-34, Eindhoven University of Technology, 1995. Available at ftp://ftp.win.tue.nl/pub/techreports/cosor/.

[12] A.W.J. Kolen, A genetic algorithm for frequency assignment, Technical report, Maastricht University, 1999.

[13] A.M.C.A. Koster, Frequency assignment—Models and algorithms, Ph.D. Thesis, Maastricht University, 1999. Available at http://www.zib.de/koster/.

[14] A.M.C.A. Koster, C.P.M. van Hoesel, and A.W.J. Kolen, The partial constraint satisfaction problem: Facets and lifting theorems, Oper Res Lett 23 (1998), 89–97.

[15] A.M.C.A. Koster, C.P.M. van Hoesel, and A.W.J. Kolen, Solving frequency assignment problems via tree-decomposition, Technical Report RM 99/011, Maastricht University, 1999. Available at http://www.zib.de/koster/.

[16] M. Padberg, The Boolean quadric polytope: Some characteristics, facets and relatives, Math Program 45 (1989), 139–172.

[17] N. Robertson and P.D. Seymour, Graph minors. II. Algorithmic aspects of tree-width, J Alg 7 (1986), 309–322.